Preprint
UCRL-JC-134499

Towards Corner Matching for Image Reconstruction

T.J. Jankun-Kelly, B. Hamann, K.I. Joy and S.P. Uselton

This article was submitted to
Institute of Electrical and Electronics Engineers Visualization '99
San Francisco, CA
October 26-30, 1999

U.S. Department of Energy
Lawrence Livermore National Laboratory

June 14, 1999

Towards Corner Matching for Image Reconstruction

T.J. Jankun-Kelly, Bernd Hamann, Kenneth I. Joy
University of California, Davis *

Samuel P. Uselton
Lawrence Livermore National Laboratory †

Abstract

A common problem in so-called multi-source visualization data analysis and visualization is the identification of certain common features in two-dimensional (2D) images and thier counterparts in three-dimensions (3D). We discuss methods to define and effectively extract features, e.g., corners or edges, from 2D images and 3D models. Work toward this goal is described and lessons learned discussed.

1 Introduction

Multi-source visualization tools currently allow scientists and engineers to display different data (simulation, physical experiment, etc.) with a single tool. For example, the VISOR tool [5] can combine a 3D model of an airplane wing with the 2D results of tests performed on that wing in real-life to produce a 3D representation of the results of those tests. Given a set of 2D images from a pressure sensitive paint (PSP) test (see Figure 4a for a typical example), each at a different camera position for a single wing, VISOR will produce a 3D model of the wing with the test results displayed. Thus, a set of images can be visualized in one image; this process is known as *image resectioning*.

Current approaches to this problem use designated reference points provided in the 2D images and the 3D model to perform the image resectioning. These points are part of the data and must be given beforehand. Our long-term goal is the development of efficient and robust feature extraction and feature matching algorithms that would allow the superpositioning of data from 2D and 3D sources with minimal human intervention. Eliminating the need to have a human specify corresponding features off-line and minimizing any human interaction on-line will greatly reduce the time required to perform multi-soruce data comparisons. This paper discusses the initial steps toward that goal.

2 Approach

Given a set of 2D images corresponding to projections of part of a 3D model, our algorithms need to find a set of points that correspond in the image and the model in order to facilitate image resectioning. The problem has three natural phases: 1) find the points of interest in the 2D images; 2) find points of interest in the 3D model; and 3) find a correspondence between the points to perform the resectioning. A solver already exists to solve the "back-projection" problem that takes the corresponding points and solves for the camera parameters that transform the 3D model into a 2D image; this projection can then be used to set texture coordinates for the 2D images. The solver requires 8 pairs of corresponding points to perform the inverse transform. Thus, the main task was to determine what points of interest to find and how to find them. Another goal is to make the process as automated as possible.

Previous work into similar problems provided little insight for this model. For example, image-based rendering (IBR) [2] attempts to display photorealistic scenes by rendering 2D images of that scene in a 3D space. Their techniques contain regularly shaped architectural models very dissimilar to the irregularly shaped aircraft wings in our motivating application. In addition, IBR techniques are generally provided a large set of images with positional and/or light intensity information, neither of which are given in this application. Ours is a more general situation.

Our algorithms focus on corner points as the feature of choice to find correspondences; psuedo-code can be found in Figure 1. The details of our algorithm are discussed below.

INPUT: Set of binary 2D images I, 3D triangulated model M, 2D corner angle δ, 3D corner angle γ.
OUTPUT: Lists F_I and F_M of feature corners in I and M respectively.
for all images $i \in I$ **do**
  Extract boundary pixels in i and put them in list B_i.
  Merge points in B_i into near-linear segments, and then attempt to merge lines with deflection angles greater than δ.
  Put the remaining points of B_i, the corners of image i, into F_I.
end for
for all vertices in $v \in M$ **do**
  Determine the set P, the palette of triangles containing v as a vertex.
  for all triangles $t_i, t_j \in P$ sharing a common edge e **do**
    Find the outside-facing unit normals $\mathbf{n_i}, \mathbf{n_j}$ of t_i, t_j respectively.
    if $\mathbf{n_i} \cdot \mathbf{n_j} \geq \gamma$ **then**
      Add e to a list of feature edges E_v.
    end if
  end for
  if $|E_v| \geq 3$ **then**
    Add v to the corner list F_M.
  end if
end for

Figure 1: Psuedo-code for our feature finding algorithm.

The first part of our algorithm focuses on the extraction of corners from 2D images. Finding corner points in 2D images is a classic computer vision and image processing problem; we used their techniques to find corners in our images [6, 3, 1]. All images are treated as binary images consisting of foreground and background values. Assuming only one object of interest exists withing the picture, we trace out the pixel boundary of that object and then decimate this boundary until only corner points remain. Pixels are decimated by fitting lines along the boundary of maximum length and then merging adjacent lines if the angle between them α is less than a given "deflection angle" δ (Figure 2). Note that the measured α is always the smaller of the two possible angles between the lines. After all applicable lines are merged, we have a polygonal representation of the original boundary whose vertices are the corner points for the image.

*Center for Image Processing and Integrated Computing, Computer Science Department, University of California, Davis, Davis, CA 95616. {kelly,hamann,joy}@cs.ucdavis.edu

†Formerly of MRJ Technologies, NASA Ames Research Center. uselton@llnl.gov

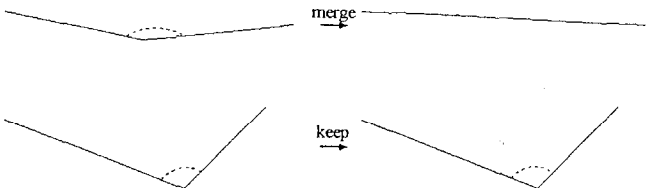


Figure 2: Line decimation in 2D corner finding.

In the second part of our algorithm, we find the feature corners of the 3D model. The extraction of corner/edge information 3D surfaces is based upon [7]. Starting with a triangular mesh (either from a directly polygonal model or a triangulation of an analytical surface), we define features of interest as follows:

Feature Edge A feature edge between to adjacent triangles t_i, t_j occurs when the angle of separation α_{ij} between the face normals $\mathbf{n_i}$ and $\mathbf{n_j}$ exceeds a threshold value γ.

Boundary Edge A boundary edge is an edge that is only part of one triangle.

Corner Point A corner occurs at a vertex where three or more feature edges meet. Optionally, any vertex on a boundary edge is a corner.

See Figure 3. For a given vertex v in the mesh, we examine each

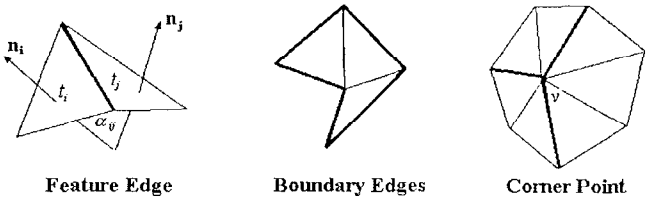


Figure 3: Feature edge and corner definitions.

edge e in the palette of triangles containing v. If two triangles share e, we compare the angle between the outward-facing normals; if the angle is greater than a specified "corner angle" γ, the edge is added to a feature edge list for that vertex. After examining all the edges, if their are three or more feature edges that contain v, we declare it a corner point. This process is repeated for each vertex in the mesh until all corners (and optionally, all feature and border edges) are extracted.

The algorithms we have developed are efficient. For the 2D case, once we have found an initial boundary pixel by some means, the algorithm in [6] is $O(B_i)$ or linear in the number of points in the boundary while the line merging is $O(B_i^2)$. For the 3D case, each vertex is examined only one while each edge is examine at most twice (once for each end-point). However, the algorithm does not always find a reasonable amount of corners. For example, if we are given a boundary or a model similar to a cirlce or sphere, there will not be enough deflection to form sharp corners. In addition, in real world images like Figure 4, some values of the program inputs can produce very large number corners that may not be found in the 3D model. Thus, some human intuition still needed in the setting of the parameters.

3 Results

The 2D algorithm was initially implemented as an extension to the VISOR problem in C++ using OpenGL at NASA Ames Research Center. The technique was later implemented in Java3D with the 3D algorithm at UC Davis.

Results of the 2D corner finding algorithm can be found in Figure 4. Various parameters can be set for the 2D algorithm, the most important being the angle of deflection that determines whether two lines form a corner or should be merged. The 24 corners found in Figure 4b have deflection angles of 135 degrees or less.

Figures 5 and 6 demonstrate the 3D corner finding algorithm; the grey spheres and white lines in the second images represent the found corners and feature/boundary edges respectively. The first model is a unit cube with various indentation and extrusions added. The displayed picture uses a corner angle of 45 degrees with results as expected. The second figure is a triangulation of a non-uniform rational B-Spline surface (NURBS) [4] with a corner angle of 23 degrees. in this example, if the angle is increased beyond 25 degrees, no corners are found; descreasing the angle produces more points clusted near the apex and nadir of the shape, but none near the flat region.

As demonstrated, the 2D corner finding algorithm is sufficient for our needs. However, the 3D corner finder is not robust enough. Though it finds corners easily for object with sharp edges (like the cube), it fails on objects with smoother variation. However, even more important, the corners found by this method may have no correspondence to the points found by the 2D algorithm. For example, for points belonging to the central indentation of Figure 5 would not be found by the 2D algorithm on that image as the are internal to the boundary. In addition, the 3D corners belonging to vertices that are obscured by faces or vertices in from of them will also not be found by the 2D algorithm since they are not visible to the camera. Finally, potential candidates for corners in 2D may not be found by the 3D algorithm since they do not occur at a vertex in the mesh. For example, the apex of Figure 6 does not occur at a vertex but may be found by the 2D algorithm as a corner. These concerns will need be addressed in the future.

4 Conculsions & Future Work

We have demonstrated a start to the problem of automated image resectioning for visualizing multi-source results. Though we can find corners in 2D, there is no clear automated solution to finding corresponding corners in 3D. Currently, our best solution is to have a user intervene and position the 3D model in such a way that our 2D algorithm can be used to find corners.

In the future, we hope to find some method for the computer to determine the positions and orientation the user uses to align the model with one of the 2D images. If some sort of efficient search of the possible model orientations can be found, then an automated solution to our problem may still be found. Using the 2D algorithm for both the model and the image has the advantage that removing corners the 2D algorithm would never find (such as those behind other objects) and finding corners that the 3D algorithm may never find (such as those caused by camera orientation). Once a reasonable 3D corner finding algorithm is realized, we will then work upon finding which found points correspond to which in the image and model.

Acknowledgments

Part of this research was supported at NASA Ames Research Center by [name the summer program]. While as UC Davis, this work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), the Office of Naval Research under contract N00014-97-1-0222, the Army Research Office under contract ARO 36598-MA-RIP, the NASA Ames Research Center through an NRA award under contract NAG2-1216, the Lawrence

Livermore National Laboratory through an ASCI ASAP Level-2 contract under W-7405-ENG-48 (and B335358, B347878), and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of Silicon Graphics, Inc., and thank the members of the Visualization Thrust at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

[1] Dana H. Ballard and Christoper M. Brown. *Computer Vision*. Prentice Hall, 1982.

[2] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Computer Graphics (SIGGRAPH '96)*, August 1996.

[3] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

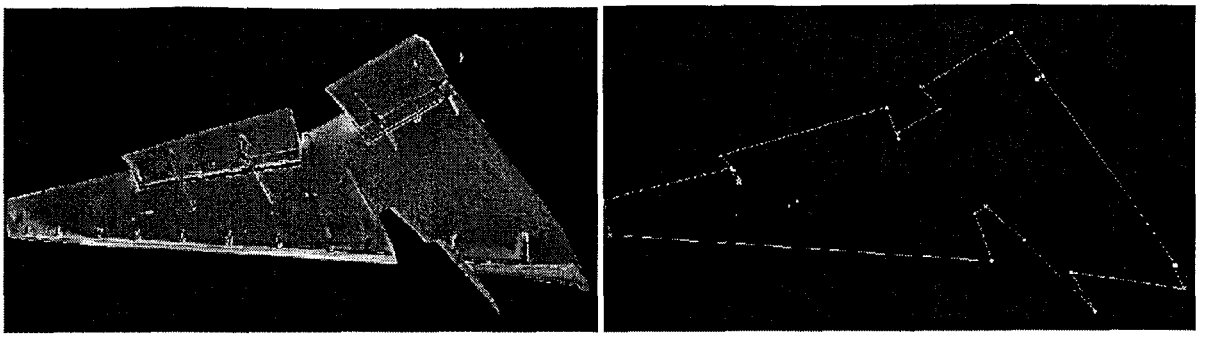
[4] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, 4th edition, 1997.

[5] Leslie Keely and Samuel P. Uselton. Development of a multisource visualization prototype. In *IEEE Visualization '98*, October 1998.

[6] Theo Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, 1982.

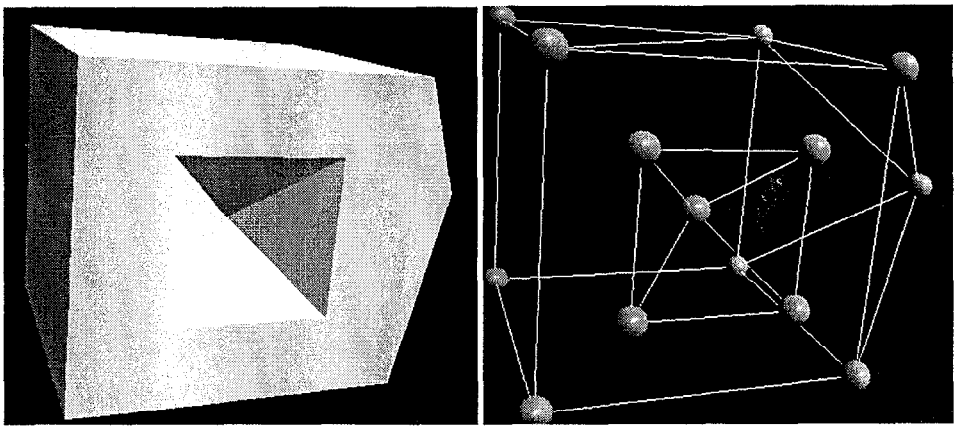
[7] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92)*, 26(2), August 1992.

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.



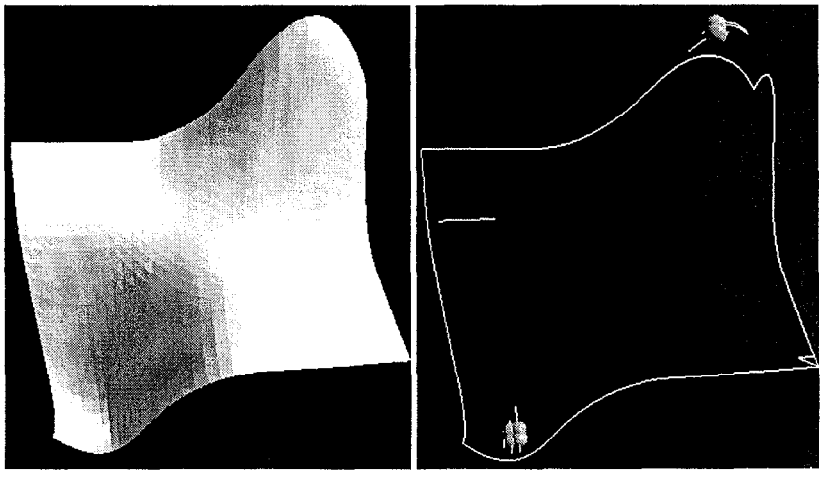
(a) The original image. (b) The extracted corners.

Figure 4: Results of 2D corner extraction algorithm on wind tunnel experiment data.



(a) The original image. (b) The extracted corners.

Figure 5: 3D corner extraction of 45 degree corners on a modified cube (16 corners).



(a) The original image. (b) The extracted corners.

Figure 6: 3D corner extration of a parametric surface (6 corners).